GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FINAL DE GRADO

Cliente de escritorio para la plataforma de teleradiología Actualpacs

---

Autor:
Carlos Castillo Melgarejo

Supervisor:
Sergio Fabra Llopis
Tutor académico:
V. Javier Traver Roig

Fecha de lectura: 29 de Septiembre de 2015

Curso académico 2014/2015

Resumen

Este proyecto ha consistido en el desarrollo de un cliente de escritorio para plataformas Windows y OS X que mediante una interfaz permita a los clientes de Actualpacs llevar a cabo el mantenimiento de estudios radiológicos actualizados en la nube siguiendo el estándar DICOM. Se han llevado a cabo las fases de análisis, diseño e implementación y pruebas propias de ingeniería de software. Mediante esta aplicación los radiólogos suben y almacenan sus estudios en la nube, conocen el estado de la base de datos, junto al estado de los envíos y recibos de estudios. Además, permite configurar de manera sencilla la compresión, cifrado y resto de opciones del sistema.

Palabras clave

Windows, OS X, Aplicación de escritorio, Radiología, Imagen médica.

Keywords

Windows, OS X, Desktop app, Radiology, Medical imaging.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Justificación y motivación del proyecto

Este proyecto ha sido desarrollado en la empresa *Actualtec Innovación Tecnológica S.L.*, una empresa de base tecnológica dedicada a proporcionar soluciones informáticas para empresas y particulares, dividida en dos líneas de negocio: *Actualweb* (servicios web) y *Actualmed* (servicios médicos).

Actualweb, ofrece servicios de Internet: diseño y programación de páginas Web, registro de dominios, alojamiento web, posicionamiento en buscadores, tiendas virtuales, copias de seguridad remotas y de almacenamiento on-line. En cambio, *Actualmed*, enfocada al campo de la medicina, ofrece soluciones óptimas en radiología y teleradiología para una mejor productividad y gestión de imágenes médicas.

Este proyecto tiene sentido ligado a las tareas realizadas en *Actualmed*, más en concreto a su producto estrella, *Actualpacs*. Es una aplicación web para el archivado y gestión de imágenes médicas, que permite a radiólogos mejorar la productividad y la calidad en el diagnóstico. Se encarga de almacenar y gestionar en la nube, imágenes médicas generadas en estudios radiológicos, además de poder acceder y visualizar toda la información de estos estudios en la web desde cualquier dispositivo.

Al inicio de la estancia en prácticas, *Actualpacs* contaba con una aplicación web que permitía el almacenamiento, gestión y visualización de estudios radiológicos en la nube. Además, unos meses antes se puso a disposición de los clientes *Actualpacsd,* un nuevo servicio que automatiza el proceso de subida de estudios a la nube.

Antes de este servicio los estudios realizados se subían manualmente mediante la aplicación web, algo que resultaba incomodo a los usuarios. Por este motivo se decidió desarrollar este nuevo servicio, el cual está compuesto por un proceso del sistema que escucha constantemente a un puerto para recoger nuevos estudios entrantes y posteriormente realizar automáticamente un proceso de compresión, cifrado y subida de los estudios a la nube siguiendo el estándar DICOM.

Actualpacsd es un gran avance, pero el hecho de ser imperceptible para el usuario, no reflejar ningún tipo de información sobre las transferencias y su dificultad para ser configurado incentivó la idea de combinar este servicio con una interfaz gráfica para crear un cliente de escritorio.

Este proyecto, abarca todo el desarrollo del cliente de escritorio *Actualpacs* que surge de la necesidad de añadir una interfaz gráfica que permita visualizar el estado de la transferencia de estudios y facilitar el procedimiento de configuración de *Actualpacsd*. Además se han añadido nuevas funcionalidades como son: Control de acceso a la aplicación, visualización de la información de los estudios recibidos y visualización de *logs* generados por *Actualpacsd*.

1.2. Objetivos del proyecto

El principal objetivo del proyecto es diseñar y desarrollar un cliente de escritorio con interfaz gráfica a partir de *Actualpacsd* que automatiza el proceso de recepción y subida de estudios a la nube. Este cliente ha de permitir a los radiólogos monitorizar el estado de la transferencia de estudios, configurar *Actualpacsd*, visualizar el estado de la base de datos y los *logs* del sistema.

Mediante este nuevo sistema se pretende:

- Facilitar una interfaz con información sobre la transferencia de estudios.
- Mejorar el procedimiento de configuración del proceso Actualpacsd.
- Acceder a la información de los estudios recibidos.
- Facilitar la visualización de los *logs* generados por el sistema.

Para desarrollar este sistema, se ha realizado un proceso íntegro de ingeniería de software que a continuación se detallará. En primer lugar, se expondrá la fase de recopilación de requisitos y análisis; en segundo lugar, se describirá la arquitectura, y el diseño a nivel de componentes; finalmente, la fase de implementación y pruebas.

1.3. Descripción del proyecto

En este proyecto se ha desarrollado un cliente de escritorio para ordenadores con sistema operativo Windows o OS X. Este cliente intercambia datos con *Actualpacsd*, un servicio del sistema desarrollado por *Actualpacs*, cuya implementación está fuera del alcance de este proyecto.

Los usuarios que van a utilizar esta cliente son radiólogos y técnicos informáticos responsables del departamento de radiología. Cada máquina encargada de realizar estudios radiológicos ha de estar conectada a un ordenador con el cliente *Actualpacs* instalado previamente y en funcionamiento.

Las funcionalidades que permite realizar el cliente *Actualpacs* a los técnicos y radiólogos son las siguientes:

- Visualizar el estado y progreso de la transferencia de estudios.
- Autenticación de usuarios para utilizar la aplicación.
- Detener o iniciar el servicio *Actualpacsd*.
- Acceder a los datos de los estudios recibidos.
- Visualizar los l*ogs* generados por *Actualpacsd*.
- Configurar el servicio *Actualpacsd*.

A la hora de aplicar los nuevos cambios en la configuración es necesario reiniciar el servicio, lo que conlleva cortar la conexión a Internet del sistema. Las demás tareas

se pueden realizar independientemente del estado del servicio, porque a pesar de que el servicio esté parado, el usuario puede acceder a la información de los estudios, l*ogs*, configuración y ver el estado de las transferencias.

Para desarrollar este sistema ha sido necesario desarrollar e implementar una base de datos que persistirá la toda la información acerca de los estudios, usuarios y configuración del sistema.

Además, se ha diseñado una interfaz gráfica agradable y fiel al aspecto corporativo de la empresa. Esta interfaz es intuitiva ya que el objetivo es facilitar la configuración y monitorización del sistema. Este es un beneficio respecto a editar manualmente los ficheros de configuración, comprobar mediante consola de comandos los l*ogs* del sistema, gestionar reglas del *firewall* para añadir *puertos* y arrancar y detener *Actualpacsd*.

Capítulo 2

Planificación del proyecto

2.1. Metodología de trabajo

Este proyecto ha sido llevado a cabo mediante el uso de una metodología ágil de desarrollo de productos software. Se ha tomado esta decisión por los siguientes motivos:

- Efectividad y flexibilidad ante cambios de requisitos.
- Metodología basada en iteraciones de una a cuatro semanas.
- Al final de cada iteración se dispone de un producto funcional.
- Evitar problemas como retrasos de tiempo y complejidad.

Entre las distintas metodologías ágiles que existen, en este proyecto se ha utilizado la que quizás sea la más conocida: *SCRUM*. Se ha elegido esta metodología ya que se adecua a las características del proyecto.

La Figura 2.1 muestra el ciclo de vida de esta metodología, donde:

- El **Product Backlog** es la pila de requisitos que debe implementar el sistema. Se compone de funcionalidades y tareas que debe realizar el programador que está en constante evolución durante todo el ciclo de vida, hasta el cierre del sistema.

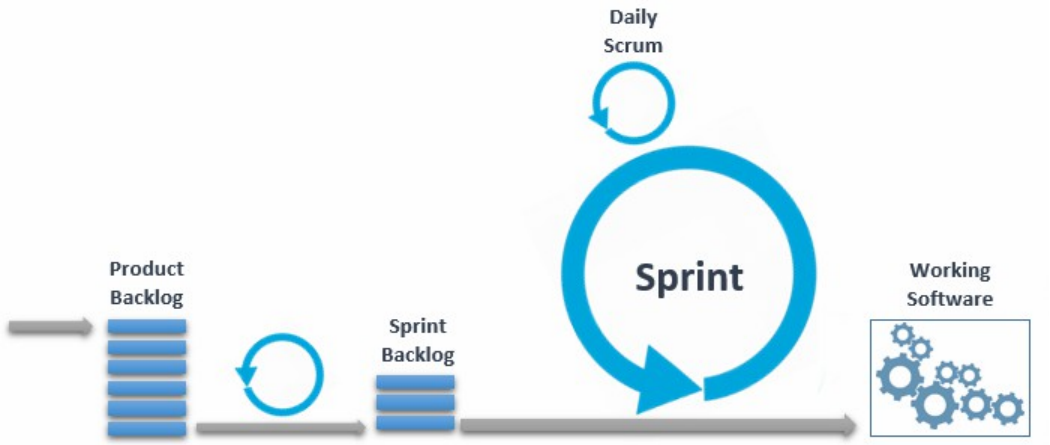

Figura 2.1: Ciclo de vida de Scrum

- El **Sprint Backlog** es la pila de tareas que se van a desarrollar en el S*print* actual. Para extraer estas tareas del P*roduct Backlog* se pueden llevar a cabo técnicas de priorización de tareas, según las necesidades del usuario y la complejidad de estas. Para llevar el seguimiento del estado de estas tareas, se ha utilizado la técnica *Kanban*, que se describe posteriormente.

- El **Sprint** es la iteración propiamente dicha, que dura de una a cuatro semanas. En esta iteración, los programadores desarrollan las tareas definidas en el S*print Backlog* y, al finalizar uno de estos, el resultado es un producto funcional que el cliente es capaz de ejecutar y tiene un valor añadido respecto a la iteración.

Además, al inicio y fin de cada iteración se realizan reuniones de la entrega y construcción del *Backlog*, e inspección del incremento integrado en cada *Sprint*. La metodología también define que se han de realizar reuniones diarias de 15 minutos aproximadamente para identificar problemas y obstáculos para resolverlos lo antes posible.

El equipo de desarrollo de *SCRUM* está formado por los siguientes roles:

- **Product Owner:** Decide la funcionalidad del producto y es el responsable de priorizar las tareas a desarrollar en cada iteración. Representa el usuario del sistema.

- **Scrum Manager:** Motiva y coordina al equipo y es responsable de detectar los problemas que pueden surgir durante el proceso.

- **Team Scrum:** Crean el producto en sí y son un equipo multidisciplinar de programadores, *testers*, analistas, etcétera.

Todos los miembros del equipo de desarrollo han de conocer con detalle la visión del P*roduct Owner* y han de colaborar regularmente y de manera directa con este.

Aunque este proyecto se ha basado en una metodología ágil, en cada iteración también se ha realizado un proceso de ingeniería de software clásico; el apartado de desarrollo de la aplicación se divide en cuatro fases bien diferenciadas:

- *Análisis de requisitos* En esta fase, se estudian las necesidades del cliente y qué espera que el sistema ofrezca. También, se identifican los distintos actores que estarán involucrados así como qué rol van a tomar en el sistema. Como resultado, se obtiene un diagrama de casos de uso, una recopilación de requisitos de usuario formalizada y una serie de prototipos sencillos de la interfaz de usuario.

- *Diseño e implementación* En esta fase tiene lugar el diseño de ofrece una visión externa del sistema, con todos los componentes que lo forman, alto nivel. Además, se realiza el diseño a nivel de componentes, diseño de clases, identificación de patrones de diseño, etc. Finalmente, consultando los prototipos obtenidos en la fase anterior, se obtiene el diseño real de la interfaz de usuario y se lleva a cabo la construcción propiamente dicha del sistema.

- *Validación y verificación* Para lanzar el sistema a producción en distintos escenarios para asegurar su correcto funcionamiento. Se han llevado a cabo pruebas unitarias, pruebas de rendimiento y pruebas de usuario para asegurar el correcto funcionamiento y calidad del sistema.

- *Despliegue o implantación* En esta fase, el producto es lanzado en un entorno de producción. En este casos, esta fase se ha dividido en dos: lanzar el producto para un grupo reducido de usuarios y estos reportar errores, en caso de encontrarlos, y finalmente, una vez el sistema funciona correctamente durante un tiempo establecido, realizar el lanzamiento masivo para todos los usuarios.

Como se ha dicho anteriormente, en este proyecto se han puesto en práctica metodologías ágiles, por tanto, el desarrollo de estas cuatro fases no ha sido lineal. Para cada iteración se ha llevado a cabo un pequeño proceso de ciclo de vida clásico formado por estas fases. De esta forma, al finalizar cada *sprint*, se obtiene una parte funcional del sistema que el usuario puede validar.

2.2. Planificación temporal

En este apartado se muestran las tareas a realizar para llevar a cabo este proyecto. Debido a que el tiempo total de estancia por parte del alumno en la empresa debe ser de unos 60 días, ya que son 300 horas a cinco horas por jornada, se han realizado las siguientes estimaciones con ayuda del supervisor de la empresa, aplicando su juicio por experiencia en el desarrollo de proyectos similares.

Se pueden observar en la Figura 2.2 estas tareas junto a un diagrama de *Gantt*, donde se muestra la duración y la disposición en el tiempo de cada una. Estas tareas son las típicas de un desarrollo ciclo de vida clásico. En cambio, el proyecto se ha desarrollado mediante una metodología ágil. En cada iteración de *SCRUM,* se han llevado a cabo un análisis, diseño e implementación de los requisitos de usuario que se pretendía desarrollar en ese periodo. Es decir, las historias de usuario escogidas de la pila del producto para esa iteración.

Se han necesitado un total de seis *Sprints* para desarrollar el producto, uno cada semana. Además, la pila del producto no ha estado definida completamente desde el principio, sino que se han ido añadiendo y cambiando historias de usuario a medida que se avanzaba.

	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto final de grado**	**60 días**	**lun 16/03/15**	**vie 05/06/15**
2	**Desarrollo propuesta técnica**	**4 días**	**lun 16/03/15**	**jue 19/03/15**
3	Definición del proyecto	1 día	lun 16/03/15	lun 16/03/15
4	Documentar el proyecto	3 días	mar 17/03/15	jue 19/03/15
5	Planificar proyecto	2 días	mar 17/03/15	mié 18/03/15
6	**Análisis de requisitos**	**5 días**	**vie 20/03/15**	**jue 26/03/15**
7	Definir casos de uso	1 día	vie 20/03/15	vie 20/03/15
8	Definir requisitos	1 día	lun 23/03/15	lun 23/03/15
9	Diseñar prototipos	3 días	mar 24/03/15	jue 26/03/15
10	Validar requisitos	0 días	jue 26/03/15	jue 26/03/15
11	**Diseño e implementación**	**34 días**	**vie 27/03/15**	**mié 13/05/15**
12	Diseño arquitectura	2 días	vie 27/03/15	lun 30/03/15
13	**Diseño a nivel de componentes**	**9 días**	**mar 31/03/15**	**vie 10/04/15**
14	Documentar arquitectura	2 días	mar 31/03/15	mié 01/04/15
15	Validar procesos	1 día	jue 02/04/15	jue 02/04/15
16	Diseñar clases de modelo	2 días	vie 03/04/15	lun 06/04/15
17	Detectar patrones	2 días	mar 07/04/15	mié 08/04/15
18	Refactorizar	2 días	jue 09/04/15	vie 10/04/15
19	Impementación	23 días	lun 13/04/15	mié 13/05/15
20	**Pruebas**	**14 días**	**jue 14/05/15**	**mar 02/06/15**
21	Detectar pruebas de unidad	2 días	jue 14/05/15	vie 15/05/15
22	Realizar y validar pruebas	4 días	lun 18/05/15	jue 21/05/15
23	Realizar pruebas de inegración	4 días	vie 22/05/15	mié 27/05/15
24	Realizar pruebas de sistema	4 días	jue 28/05/15	mar 02/06/15
25	**Puesta en marcha**	**3 días**	**mié 03/06/15**	**vie 05/06/15**
26	Preparación de la instalación	2 días	mié 03/06/15	jue 04/06/15
27	Redactar manual de usuario	2 días	mié 03/06/15	jue 04/06/15
28	Implantación	1 día	vie 05/06/15	vie 05/06/15

Figura 2.2: Tareas y diagrama de Gantt

2.3. Desviación de la planificación inicial

Durante la estancia de prácticas en la empresa, se decidió realizar un cambio en el horario de la jornada laboral, y, a partir de la 3 semana se pasó a realizar 8 horas al día en lugar de 5. Este cambio afectó a la planificación inicial del proyecto, adelantando la fecha de finalización de la estancia en prácticas del 5 de junio al 19 de mayo, lo cual supuso la necesidad de realizar una nueva planificación.

En la Figura 2.3 se puede observar esta nueva planificación del proyecto. El cambio a jornada completa se efectúa el día 6 de abril, con lo cual se obtienen 13 días con una jornada de 5 horas y 30 días a jornada completa, sumando así un total de 305 horas de estancia en la empresa.

Finalmente, todas las tareas y objetivos definidos al inicio del proyecto se han alcanzado en el tiempo global esperado y los resultados han sido satisfactorios. Además, todos los requisitos iniciales del cliente se han alcanzado antes de lo planificado, por lo tanto se ha decidido añadir nuevos requisitos al proyecto a medida que iba creciendo.

	Nombre de tarea	Duración	Comienzo	Fin
1	**Proyecto final de grado**	**43 días**	**lun 16/03/15**	**mar 19/05/15**
2	**Desarrollo propuesta técnica**	**4 días**	**lun 16/03/15**	**vie 20/03/15**
3	Definición del proyecto	1 día	lun 16/03/15	lun 16/03/15
4	Documentar el proyecto	3 días	mar 17/03/15	vie 20/03/15
5	Planificar proyecto	2 días	mar 17/03/15	mié 18/03/15
6	**Análisis de requisitos**	**5 días**	**lun 23/03/15**	**vie 27/03/15**
7	Definir casos de uso	1 día	lun 23/03/15	lun 23/03/15
8	Definir requisitos	1 día	mar 24/03/15	mar 24/03/15
9	Diseñar prototipos	3 días	mié 25/03/15	vie 27/03/15
10	Validar requisitos	0 días	vie 27/03/15	vie 27/03/15
11	**Diseño e implementación**	**25 días**	**lun 30/03/15**	**mié 06/05/15**
12	Diseño arquitectura	1 día	lun 30/03/15	lun 30/03/15
13	**Diseño a nivel de componentes**	**9 días**	**mar 31/03/15**	**mar 14/04/15**
14	Documentar arquitectura	2 días	mar 31/03/15	mié 01/04/15
15	Validar procesos	1 día	jue 02/04/15	jue 02/04/15
16	Diseñar clases de modelo	2 días	mar 07/04/15	mié 08/04/15
17	Detectar patrones	2 días	jue 09/04/15	vie 10/04/15
18	Refactorizar	2 días	lun 13/04/15	mar 14/04/15
19	Impementación	15 días	mié 15/04/15	mié 06/05/15
20	**Pruebas**	**7 días**	**jue 07/05/15**	**vie 15/05/15**
21	Detectar pruebas de unidad	1 día	jue 07/05/15	jue 07/05/15
22	Realizar y validar pruebas	2 días	vie 08/05/15	lun 11/05/15
23	Realizar pruebas de inegración	2 días	mar 12/05/15	mié 13/05/15
24	Realizar pruebas de sistema	2 días	jue 14/05/15	vie 15/05/15
25	**Puesta en marcha**	**2 días**	**lun 18/05/15**	**mar 19/05/15**
26	Preparación de la instalación	1 día	lun 18/05/15	lun 18/05/15
27	Redactar manual de usuario	1 día	lun 18/05/15	lun 18/05/15
28	Implantación	1 día	mar 19/05/15	mar 19/05/15

Figura 2.3: Nueva planificación

Capítulo 3

Entorno tecnológico

Este proyecto se ha desarrollado con la intención de poder utilizar el cliente desarrollado tanto en plataformas Windows y MacOSX en el lenguaje *C++*. Se han utilizado un conjunto de herramientas y tecnologías, cada una de ellas para un propósito concreto.

Las herramientas y tecnologías utilizadas son las siguientes:

- *Qt*: Amplia plataforma de desarrollo que incluye clases, librerías y herramientas para la producción de aplicaciones con interfaz gráfica en *C++* que pueden operar en varias plataformas. Con *Qt* se pueden desarrollar ricas aplicaciones gráficas, incluye soporte de nuevas tecnologías como *OpenGL*, XML, bases de datos, programación para redes y mucho más.

- *DCMTK*: Colección de librerías y aplicaciones que implementan partes del estándar DICOM. DCMTK incluye el software necesario para el examen, la construcción y la conversión de archivos de imagen DICOM, manejando los medios de comunicación, enviando y recibiendo imágenes sobre la conexión *network*, así como la demostración de almacenaje de imágenes y bases de datos.

- *SQLite*: La aplicación necesita persistencia local, y el sistema de gestión de bases de datos que se ha usado ha sido *SQLite* por su sencilla integración con la plataforma *Qt*.

- *Boost*: Para evitar errores de compatibilidad dependiendo de la plataforma en la que se ejecute el proyecto, se incluye la biblioteca *Boost* que proporciona métodos y herramientas multiplataforma que no proporciona *Qt*.

En las siguientes secciones se explica con profundidad cada una de estas tecnologías y herramientas.

3.1. Qt

Qt se diferencia de una librería *C++* cualquiera puesto que añade muchísimas funcionalidades a *C++*, cambiándolo de tal forma, que prácticamente crea un nuevo lenguaje de programación. Además, facilita la programación de entornos gráficos respecto a *C++*, cosa que favorece al desarrollo del proyecto.

Otro punto clave de *Qt* es que se considera una biblioteca multiplataforma, ya que permite programar el mismo código independientemente del sistema operativo en el que vaya a ser utilizado. Una ventaja frente a *Java* es la eficiencia gracias a no necesitar máquina virtual de por medio, ya que el compilador se encarga de compilar el código expresamente para cada máquina.

Qt fue desarrollada inicialmente por la empresa noruega Trolltech y posteriormente comprada por Nokia en 2008. Existe en tres tipos de licencia disponibles para esta librería: GNU LGPL, GNU GPL y Propietaria (Nokia). Además, ha sido utilizado para desarrollar software de todos los ámbitos como por ejemplo: *Skype, Mathematica, Google Earth, Adobe Photoshop Album, Dropbox,* etc.

3.1.1 El modelo de objetos Qt

Qt tiene un modelo de objetos muy interesante. Esta arquitectura es la que hace que sea tan potente y fácil de usar. Los dos pilares de *Qt* son la clase *QObject* (objeto *Qt*) y la herramienta MOC (MetaObjects Compiler).

La programación mediante el modelo de objetos *Qt* se fundamenta en derivar las clases nuevas que se creen de objetos *QObject*. Al hacer esto, se heredan una serie de propiedades que caracterizan a *Qt* (y lo hacen especial frente al *C++* estándar):

- Gestión simple de la memoria.
- *Signals* y *slots*.
- Propiedades.
- Auto-conocimiento.

No hay que olvidar, sin embargo, que *Qt* no deja de ser *C++* estándar con macros, por lo que cualquier aplicación programada en *Qt*, puede tener código en *C++* estándar o incluso en *C*.

De estas características, se comentarán en los siguientes apartados las dos primeras, por ser las empleadas principalmente en el proyecto y en general en cualquier aplicación escrita en *Qt*.

3.1.1.1 Gestión simple de la memoria

Cuando se crea una instancia de una clase derivada de un *QObject* es posible pasarle al constructor un puntero al objeto padre. Esta es la base de la gestión simple de memoria.

Al borrar un objeto padre, se borran todos sus objetos hijos asociados. Esto quiere decir que una clase derivada de *QObject* puede crear instancias de objetos "hijos-de-QObject" pasándole el puntero como padre sin preocuparse de la destrucción de estos hijos.

Esto es una gran ventaja frente al *C++* estándar, ya que en *C++* estándar cuando se crea un objeto siempre hay que tener cuidado de destruir manualmente los objetos que lo componen uno a uno para liberar memoria, cosa que *Qt* hace automáticamente eliminando solo al objeto padre.

Figura 3.1: Ejemplo de Qwidget padre con hijos

Un ejemplo de aplicación podría ser por ejemplo crear una ventana *QWidget* con dos botones *QPushButton* y una entrada de texto *QlineEdit* (Figura 3.1). La clase *Qwidget* deriva de *Qobject* y contiene dentro los elementos gráficos *QPushButton* y *QLineEdit* . Por tanto, estos elementos gráficos son hijos de la clase *Qwidget.*

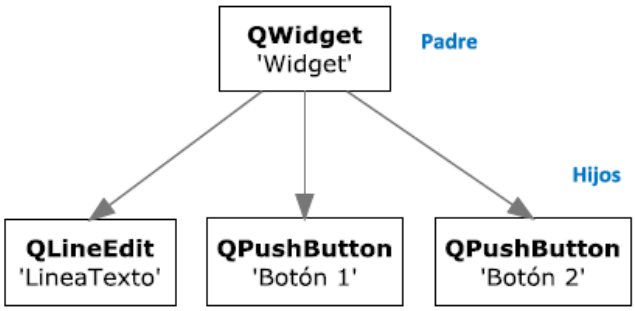

Figura 3.2: Jerarquía de memoria del QWidget de la figura 3.1.

Al destruir la ventana se destruyen los hijos liberando la memoria correspondiente de manera transparente para el programador. La jerarquía de memoria se puede ver en la figura 3.2. Cabe destacar que de cada *QObject* también pueden crearse objetos hijos, aumentando el árbol de memoria. Estos objetos "nietos" del padre original también se eliminan al eliminarse su objeto padre.

La terminología de hijos y padres no debe confundirse con la utilizada en los procesos, ya que cuando se habla de padres e hijos, nos estamos refiriendo a objetos *Qt.*

3.1.1.2 Signals y slots

Las *signals* y los *slots* (señales y ranuras, en castellano) son los que hacen que los diferentes componentes de *Qt* sean tan reutilizables. Proporcionan un mecanismo a través del cual es posible exponer interfaces que pueden ser interconectadas libremente (figura 3.3).

Un ejemplo de una *signal* podría ser la señal que se emite al pulsar un botón. Por otro lado. un *slot* podría ser por ejemplo, un observador que espera recibir esta señal para realizar una serie de tareas que le hayan sido asignadas.

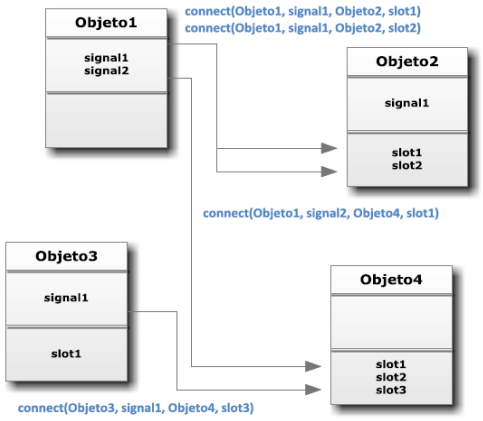

Figura 3.3: Conexión entre signals y slots de diferentes objetos.

La ventaja principal de las *signals* y de los *slots* es que el *QObject* que envía la señal no tiene que saber nada del objeto receptor. Este tipo de diseño se llama en inglés *loose coupling* y permite integrar muchos componentes en el programa de manera sencilla y minimizando la probabilidad de errores.

Para poder usar las *signals* y los *slots* tienen que ser declarados en un fichero de cabecera (normalmente *.h* o *.hpp*) y ser implementados en un fichero fuente (normalmente *.cpp*). Este fichero de cabecera a la hora de compilar pasa a través de una herramienta conocida como el MOC (compilador de metaobjetos). El MOC produce un *.cpp* complementario que contiene el código que permite que funcionen las *signals* y los *slots* y otras muchas características de *Qt*.

En la figura 3.4 se puede ver el flujo desde que se crea una clase derivada de *QObject* hasta que se compila:

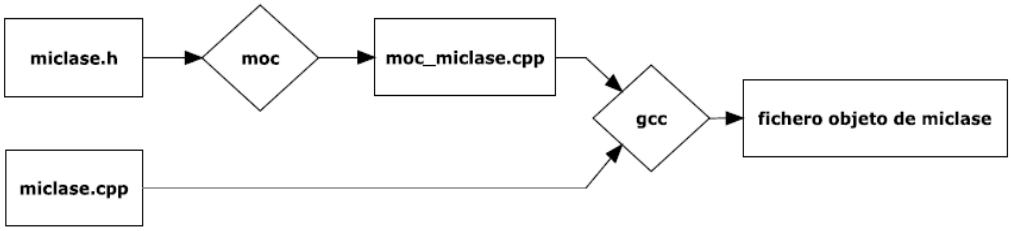

Figura 3.4: Flujo del MOC.

Aun así, como ya se ha mencionado en la introducción de este capítulo, una aplicación *Qt* sigue siendo *C++*. Por tanto, las *signals* y los *slots* son simplemente palabras reservadas o *keywords* que reemplaza el preprocesador convirtiéndolas en código *C++* real.

Los *slots* se implementan como cualquier método de la clase, mientras que las *signals* las implementa el MOC. Cada objeto tiene una lista de conexiones (qué *slots* se activan con qué *signals*) y *slots* (cuáles se usan para construir la tabla de conexiones en el método *connect*). Las declaraciones de estas tablas están escondidas en la macro *Q_OBJECT*. Esto se puede ver en el código de la figura 3.5.

```
class MiObjeto : public QObject
{
  Q_OBJECT
public:

  MiObjeto( QObject *parent=0) : QObject( parent)
  {
    // . . .
  }

public slots:
  void miSlot( void )
  {
    // . . .
  }
signals:
  void miSignal();
};
```

Figura 3.5: Código fuente genérico de un Qobject.

3.1.2 Qt Creator

Qt Creator es un IDE (Entorno Integrado de Desarrollo) multiplataforma creado inicialmente por Trolltech y mantenido en la actualidad por Nokia. Requiere como mínimo la versión 4 de *Qt*. Integra un editor de texto, un depurador y *Qt Designer*.

Qt Designer es una herramienta muy potente (antes era un programa independiente), utilizada en varias partes de este proyecto, que permite crear *widgets* o elementos gráficos de manera visual, facilitando en gran medida la creación de código para la parte de la GUI. Tras diseñar de manera gráfica un *widget*, se genera un archivo con extensión *.ui*, que no es más que un archivo XML que posteriormente se traduce a una clase en *C++* mediante el UIC (User Interface Compiler).

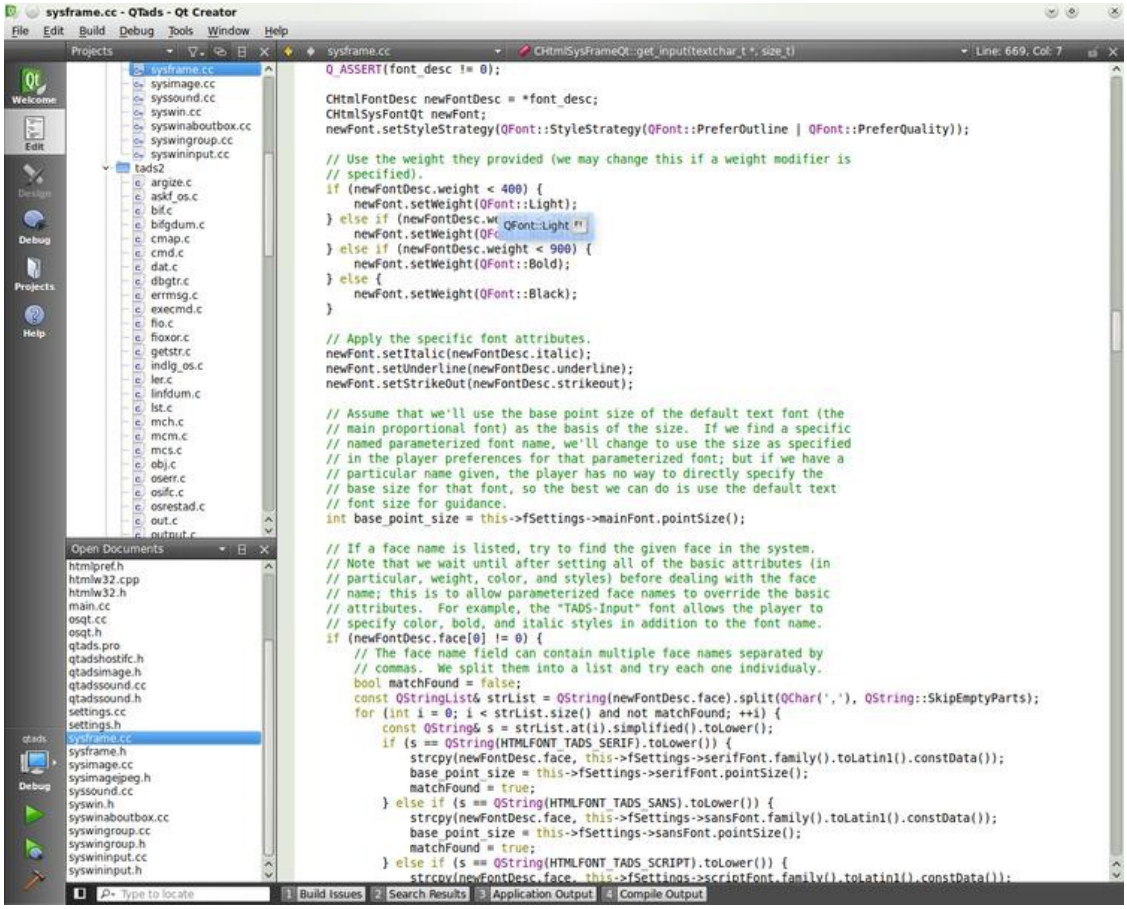

Figura 3.6: Qt Creator

3.2. DCMTK

DCMTK es una colección de librerías y aplicaciones que implementan las partes del estándar DICOM para la comunicación de imagen médica. Esto incluye el software para el examen, la construcción y la conversión archivos de imagen DICOM, el manejo medios de comunicación autónomos, el envío e imágenes de encubrimiento sobre una conexión de red, así como la demostración del almacenaje de imágenes y servidores *worklist*. DCMTK incluye el código fuente completo y está escrito en una mezcla de *C ANSI* y *C ++*.

DCMTK es usado por hospitales y empresas en todo el mundo para una amplia variedad de propósitos desde ser utilizado como una herramienta para pruebas de productos, a ser un componente básico para proyectos de investigación, prototipos y productos comerciales

El software DCMTK puede ser compilado bajo Windows NT o una amplia gama de sistemas operativos Unix que incluyen Linux, Solaris, OSF/1, IRIX, FreeBSD y MacOS X. Todos los guiones de configuración necesarios y ”makefiles” de los proyecto son suministrados.

Trabajar con esta herramienta es muy arduo y dificultoso debido a la poca documentación, a la dificultad que entraña el trabajar con todos los términos DICOM y a la amplitud del código. Por estas razones y por alguna más se aconsejable trabajar con otras herramientas que cuentan con la desventaja de no ser gratuitas, pero con las ventajas de una gran documentación plenamente estructurada y el trabajar con Java.

3.3. SQLite

La aplicación precisa de un sistema de persistencia local. Para ello, se ha elegido el sistema de gestión de bases de datos relacional *SQLite* por diversos motivos:

1. *SQLite* se integra sin problemas con *Qt* y el consumo de memoria en tiempo de ejecución es mínimo. Es necesario optimizar los recursos de la aplicación, ya que el procesos de comprobación de nuevos estudios consume bastantes recursos.

2. La base de datos no es un proceso independiente con el que el programa principal se comunica, sino que todas las estructuras que la definen (tablas, vistas, etc...) están alojadas en un fichero.

3. Para realizar una copia de seguridad, basta con hacer una copia del fichero de la base de datos.

4. La librería de *SQLite* se integra con el mismo programa, es decir, no existe comunicación entre procesos (programa principal y base de datos), lo que reduce la latencia entre las operaciones sobre los datos almacenados.

Por otra parte, existen desventajas respecto a otros SGBD (Sistema Gestor de Bases de Datos). Por ejemplo, no existe la gestión de usuarios, por lo que la seguridad delega en el sistema de permisos de ficheros del sistema operativo. Además, tampoco permite concurrencia de conexiones ya que el fichero de la base de datos se bloquea mientras un usuario está realizando alguna modificación.

A pesar de estas desventajas, se ha considerado que el SGBD *SQLite* es más que adecuado en el contexto de este proyecto.

3.4. Boost

El diseño e implementación de la librería *Boost* permite que sea utilizada en un amplio espectro de aplicaciones y plataformas. Abarca desde bibliotecas de propósito general hasta abstracciones del sistema operativo. Con el objetivo de alcanzar el mayor rendimiento y flexibilidad se hace un uso intensivo de plantillas. *Boost* ha representado una fuente de trabajo e investigación en programación genérica y metaprogramación en *C++*.

Varios fundadores de *Boost* pertenecen al Comité ISO de Estándares *C++*. La próxima versión estándar de *C++* incorporará varias de estas bibliotecas.

Actualmente *Boost* está formada por más de 80 bibliotecas individuales, incluidas las bibliotecas de álgebra lineal, la generación de números pseudoaleatorios, multihilos, procesamiento de imágenes, expresiones regulares, pruebas unitarias, entre otros. La mayoría de las bibliotecas *Boost* están basadas en cabeceras, funciones en línea y plantillas, por lo que no tienen que ser construidas antes de su uso

Boost ha sido necesarios en este proyecto para utilizar funciones multiplataforma que *Qt* no proporciona. Concretamente, estas funciones se han utilizado para acceder a recursos del sistema como servicios, *firewall* y variables de entorno, independientemente de la plataforma donde se esté utilizando. Gracias a esto, se ha evitado escribir código distinto para cada plataforma.

Capítulo 4

Desarrollo de la aplicación

En los siguientes apartados se va a especificar la recopilación de requisitos de usuario, seguido del análisis del entorno y actores involucrados, diseño a nivel de arquitectura, componentes e interfaz gráfica y, finalmente, la implementación propiamente dicha del software.

4.1. Análisis

En esta fase se define formalmente todos los elementos que constituyen el contexto del problema. Para ello, se va a presentar los requisitos funcionales del sistema y, como consecuencia de estos, los requisitos de software y hardware. Previamente, se mostrará un diagrama de casos de uso para apreciar qué actores realizan qué determinadas acciones sobre el sistema.

4.1.1. Casos de uso

Los casos de uso son una herramienta para el análisis de proyectos software. Haciendo uso de ellos se puede observar gráficamente la interacción entre los actores involucrados y el propio sistema. En este proyecto, se pueden identificar 2 actores: El usuario de la aplicación y el servicio del sistema. El último no interactúa

directamente con la aplicación desarrollada pero es necesario tenerlos en cuenta para comprender el funcionamiento del software que se está analizando.

La Figura 4.1 muestra el diagrama de casos de uso, con los actores y las acciones que realizan sobre el sistema.

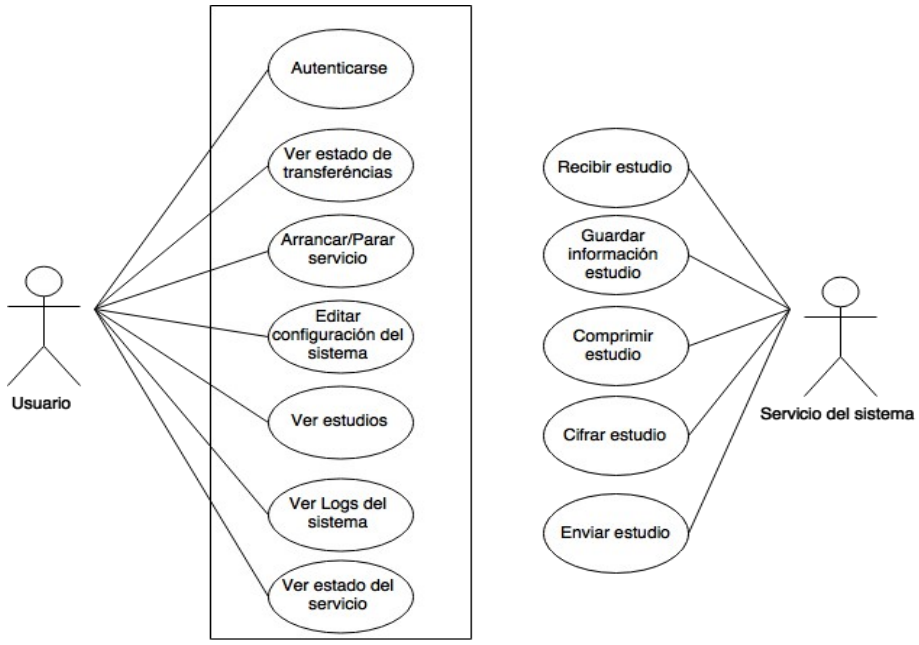

Figura 4.1: Casos de uso

Como se puede apreciar, se muestran los dos actores mencionados anteriormente, relacionados con las acciones que realiza cada uno sobre el sistema. El rectángulo que se puede observar representa el alcance del sistema, es decir, aquellas acciones que se realizan dentro del contexto de la aplicación.

4.1.2. Requisitos funcionales

El objetivo de los requisitos funcionales de usuario es definir el alcance del sistema, es decir, las características, a nivel funcional, que debe satisfacer el software. La definición formal de estos requisitos es necesaria para comprender el problema a resolver.

A continuación, se muestran las tablas que definen estos requisitos. Cada requisito contiene un identificador, nombre, descripción y una secuencia de pasos que definen su ciclo de vida.

Requisito funcional	
Código	FR01
Nombre	Autenticarse
Descripción	El sistema ha de permitir a los usuarios autenticarse para acceder a este. Si el acceso es satisfactorio, el sistema guarda los datos para realizar este proceso automáticamente las próximas veces.
Secuencia normal	
1	Introducir usuario.
2	Introducir contraseña.
3	Seleccionar opción de aceptar.
Importancia	Media
Comentarios	El sistema debe mostrar un mensaje de error en caso de que los datos introducidos sean erróneos.

Requisito funcional	
Código	FR02
Nombre	Ver estado de transferencias
Descripción	Al acceder al sistema, este debe proporcionar la información de las transferencias, tanto las de envío como las de recepción.
Importancia	Alta
Comentarios	Ninguno.

Requisito funcional	
Código	FR03
Nombre	Ver estado del servicio
Descripción	Al acceder al sistema, este debe mostrar si el servicio encargado de enviar, recibir, cifrar y comprimir los estudios, está encendido o apagado.
Importancia	Media
Comentarios	Ninguno.

Requisito funcional	
Código	FR04
Nombre	Arrancar servicio
Descripción	El usuario ha de ser capaz de arrancar el servicio encargado de enviar, recibir, cifrar y comprimir los estudios.
Secuencia normal	
1	Pulsar botón de arrancar o parar servicio.
2	Esperar a que se complete el proceso.
Importancia	Media
Comentarios	El sistema debe mostrar un mensaje de error en caso de que el proceso no se realice satisfactoriamente.

Requisito funcional	
Código	FR05
Nombre	Parar servicio
Descripción	El usuario ha de ser capaz de parar el servicio encargado de enviar, recibir, cifrar y comprimir los estudios.
Secuencia normal	
1	Pulsar botón de arrancar o parar servicio.
2	Esperar a que se complete el proceso.
Importancia	Media
Comentarios	El sistema debe mostrar un mensaje de error en caso de que el proceso no se realice satisfactoriamente.

Requisito funcional	
Código	FR06
Nombre	Editar configuración del sistema
Descripción	El sistema ha de permitir a los usuarios editar la configuración del servicio. Además si lo requiere ha de reiniciar el servicio o modificar una regla del firewall.
Secuencia normal	
1	Modificar los campos que se necesiten cambiar.
2	Presionar el botón de guardar cambios.
Importancia	Alta
Comentarios	El sistema debe mostrar un mensaje de confirmación antes de aplicar los cambios.

Requisito funcional	
Código	FR07
Nombre	Reiniciar servicio
Descripción	Después de aplicar los cambios de la configuración, el sistema debe reiniciar automáticamente el servicio para que estos cambios se efectúen.
Importancia	Alta
Comentarios	El sistema debe mostrar el progreso del proceso.

Requisito funcional	
Código	FR08
Nombre	Actualizar regla firewall
Descripción	Si a la hora de editar la configuración del servicio se cambia el puerto de recepción, el sistema ha de cambiar automáticamente la regla de firewall con el nuevo valor del puerto.
Secuencia normal	
1	Cambiar el puerto de recepción de estudios.
2	Pulsar el botón de aplicar cambios.
3	Esperar a que se complete el proceso.
Importancia	Alta
Comentarios	Ninguno.

Requisito funcional	
Código	FR09
Nombre	Ver estudios
Descripción	El sistema ha de permitir al usuario ver los estudios almacenados en la base de datos junto a las series de estos estudios.
Secuencia normal	
1	Seleccionar pestaña de estudios.
2	Pulsar sobre un estudio para desplegar sus series.
Importancia	Media
Comentarios	Los estudios han de poder ordenarse ascendente y descendentemente en cada uno de sus campos.

Requisito funcional	
Código	FR10
Nombre	Buscar estudios
Descripción	El sistema ha de permitir al usuario buscar estudios por nombre de paciente o código del estudio.
Secuencia normal	
1	Seleccionar pestaña de estudios.
2	Seleccionar sobre que campo quieres realizar la búsqueda.
3	Introducir el nombre de paciente o código de estudio que desees buscar.
4	Presionar Enter o Buscar.
Importancia	Media
Comentarios	Ninguno.

Requisito funcional	
Código	FR11
Nombre	Ver Logs
Descripción	El sistema ha de permitir al usuario ver los Logs generados por el servicio.
Secuencia normal	
1	Seleccionar pestaña de Logs.
2	Seleccionar el archivo que se quiere visualizar.
Importancia	Media
Comentarios	Ninguno.

Requisito funcional	
Código	FR12
Nombre	Filtrar Logs
Descripción	El sistema ha de permitir al usuario filtrar los Logs generados por el servicio.
Secuencia normal	
1	Seleccionar la pestaña de Logs.
2	Introducir campo a filtrar.
3	Presiona Enter o Filtrar.
Importancia	Media
Comentarios	Ninguno.

4.1.3. Requisitos de datos

Para satisfacer los requisitos de usuario expuestos anteriormente, es necesario definir las entidades y atributos que se van a almacenar, es decir, los requisitos de datos.

A continuación, se muestran las tablas que representan cada entidad y los propiedades que contendrán cada una de estas:

Requisito de datos	
Código	DR01
Nombre	Usuario
Datos	Para cada usuario el sistema debe almacenar el nombre de usuario, su contraseña cifrada.
Comentarios	Ninguno.

Requisito de datos	
Código	DR02
Nombre	Estudio
Datos	Para cada estudio el sistema debe almacenar una serie de parámetros DICOM: StudyInstanceUid, nombre e Id del paciente, Accesion number modalidad del estudio, fecha del estudio y fecha de creación del estudio.
Comentarios	Ninguno.

Requisito de datos	
Código	DR03
Nombre	Serie
Datos	Para cada serie el sistema debe almacenar el Id del estudio al que pertenece, SeriesInsanceUid, modalidad y fecha de creación.
Comentarios	Ninguno.

Requisito de datos	
Código	DR04
Nombre	Envío
Datos	Cada envío contiene el nombre del paciente, la modalidad, fecha de inicio,imágenes enviadas y imágenes totales.
Comentarios	Ninguno.

Requisito de datos	
Código	DR05
Nombre	Recepción
Datos	Cada recepción contiene el nombre del paciente, modalidad, fecha de inicio, imágenes recibidas y fecha de la última imágen recibida. contraseña.
Comentarios	Ninguno.

Requisito de datos	
Código	DR06
Nombre	Configuración
Datos	El sistema debe almacenar una serie de parámetros DICOM los cuales son los siguientes: Server URL/IP, localPacs URL/IP, server port, node port,localPacs port, node, node send, cloud, localPacs, aetitles, ACSE timeout, DIMSE timeout, connect. timeout, DIMSE blocking, connection blocking, max. associations, active node, active compresion, active send, active query retrieve, active localpacs query, rotative log, active delete, active recovery, core-multiplicator, delete time, log level, sleep during pacs request, day search studies y certificados y claves del usuario y servidor.
Comentarios	Ninguno.

4.1.4. Requisitos de software y hardware

La aplicación que se describe en este proyecto ha sido testada sobre varios ordenadores con sistema operativo Windows XP y Windows 7. Los requisitos para ejecutar esta aplicación son los siguientes:

- Ordenador con sistema operativo Windows superior o igual a XP.
- Procesador Dual Core o superior.
- Conexión a Internet para funcionalidades como, transferencia, sincronización de datos, etc.

- Almacenamiento interno mínimo para almacenar la base de datos local y almacenar estudios. El espacio dependerá de la cantidad de estudios que realice la clínica que utilice el sistema.

4.1.5. Prototipos de la interfaz

Para terminar con la fase del análisis, se han creado prototipos de las interfaces de usuario para las funcionalidades más significativas de la aplicación: *Login*, transferencia, ver *logs*, ver estudios y configuración. Estos prototipos no representan el aspecto definitivo de las pantallas, sino que su objetivo es plasmar una idea más visual de la aplicación para permitir la verificación de las funcionalidades, sin entrar en aspectos de diseño.

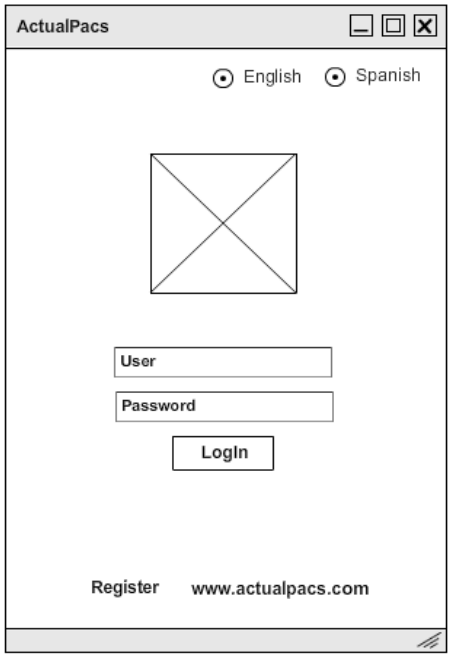

Figura 4.2: Prototipo: Login

Las figuras 4.2, 4.3, 4.4, 4.5 y 4.6 muestran los prototipos generados para las pantallas de *Login*, transferencia, ver *logs*, ver estudios y configuración, respectivamente.

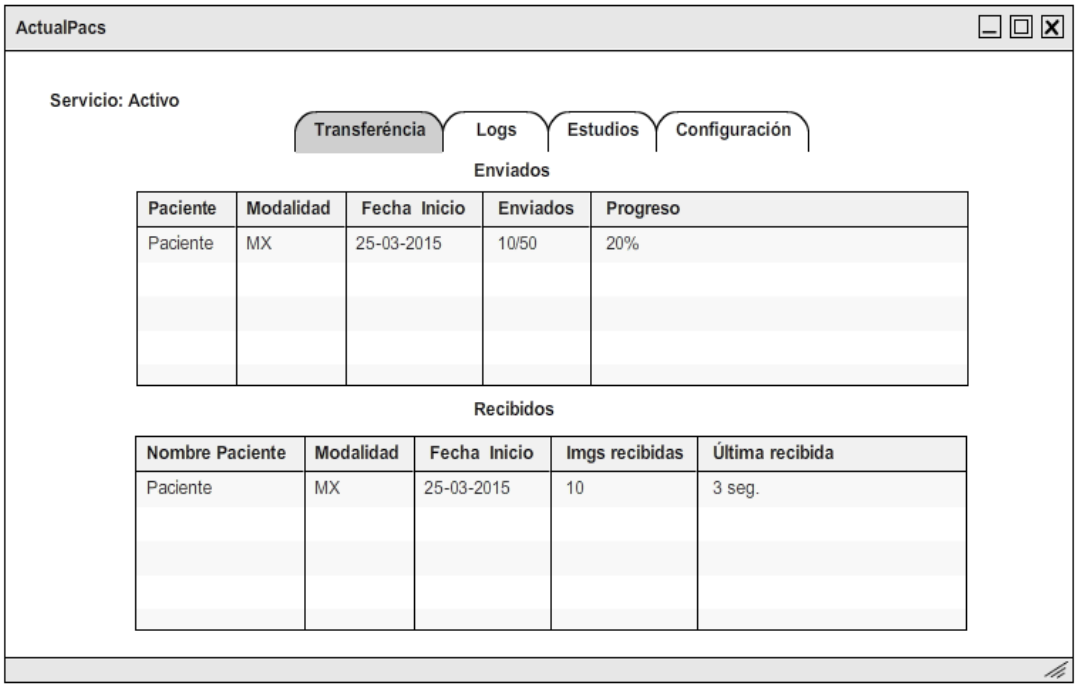

Figura 4.3: Prototipo: Transferencia

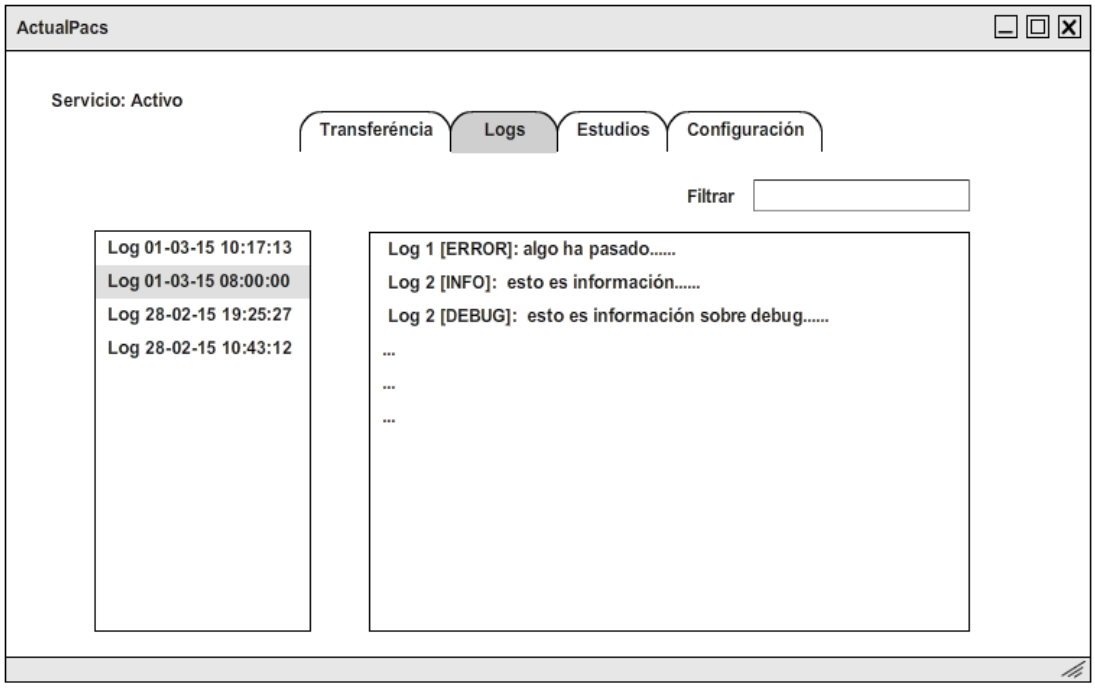

Figura 4.4: Prototipo: ver Logs

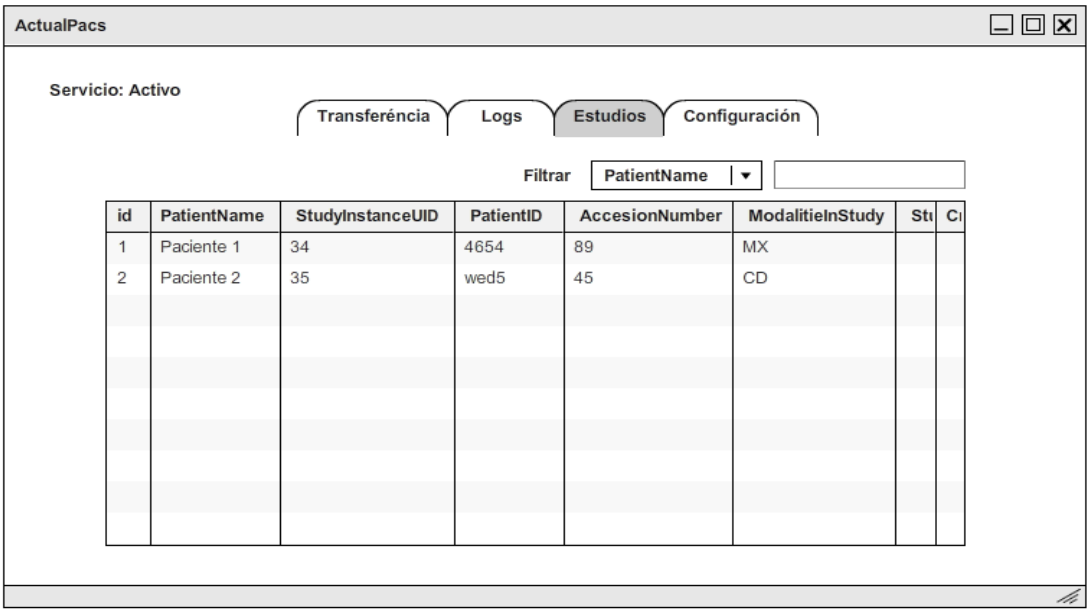

Figura 4.5: Prototipo: ver estudios

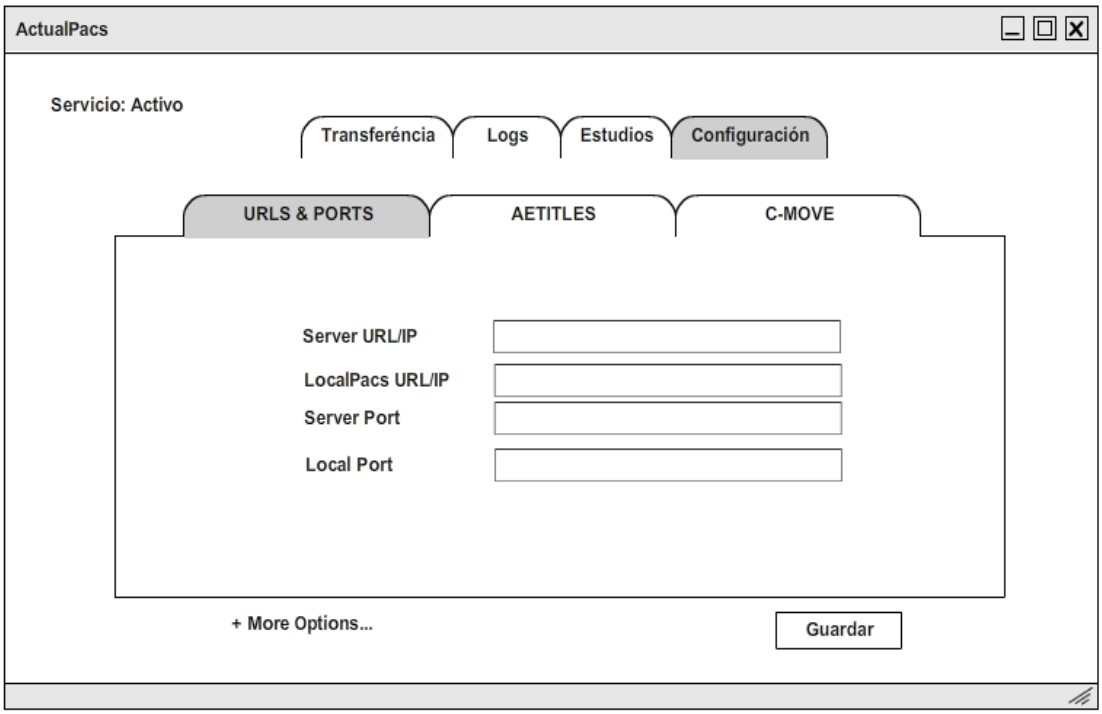

Figura 4.6: Prototipo: Configuración

4.2. Diseño e implementación

Una vez terminada la fase de análisis, se procede a modelar el sistema a nivel de arquitectura y componentes de software. La etapa de diseño permite a los desarrolladores tener una idea más clara y concreta de cómo será el sistema, reduciendo la abstracción respecto a la fase anterior.

4.2.1. Arquitectura de la aplicación

El primer paso en la fase de diseño es definir la arquitectura del sistema a alto nivel. La arquitectura es una especificación de la estructura a nivel global del sistema. Se centra en aspectos de más alto nivel que los algoritmos, funciones o tipos de datos. Estos últimos forman parte del diseño a nivel de componentes.

En este caso, la arquitectura presente se puede definir como la mezcla de dos arquitecturas: cliente/servidor y peer to peer. Existe un cliente (aplicación desarrollada) que envía datos al servidor central y a la misma vez actúa como servidor para recibir estudios de las máquinas radiológicas.

La Figura 4.7 muestra todos los módulos que están dentro del alcance así como los que no lo están pero cumplen alguna función en todo el ciclo de vida de la transferencia de un estudio.

En este diagrama encontramos cuatro módulos distintos:

- **Base de datos central**: Esta es la base de datos central de la empresa. Contiene la información de usuarios, estudios, etc.

- **Servidor central**: Esta es la máquina que se encargará de recibir y almacenar los estudios enviados por el cliente de escritorio. Además proporciona una aplicación web para el cliente, pero esto no entra dentro del alcance de este proyecto.

- **Cliente de escritorio**: Este módulo, junto con la base de datos local, componen el sistema que se está diseñando en este proyecto. El cliente recibe estudios de la máquina radiológica, comprime cifra los estudios y los sube al servidor central.

- **Cliente web**: Esta parte del sistema permite visualizar los estudios a partir de una aplicación web proporcionada por el servidor central, además de la gestión de estos mismos. Esta parte no entra en el alcance del proyecto.

- **Base de datos local**: Como se ha explicado anteriormente, la aplicación persiste datos localmente. Para ello, se ha utilizado el sistema de gestión de bases de datos *SQLite*.

- **Máquina radiológica**: Esta es la máquina que se encargará de realizar los estudios radiológicos y enviarlos a el cliente *Actualpacs*.

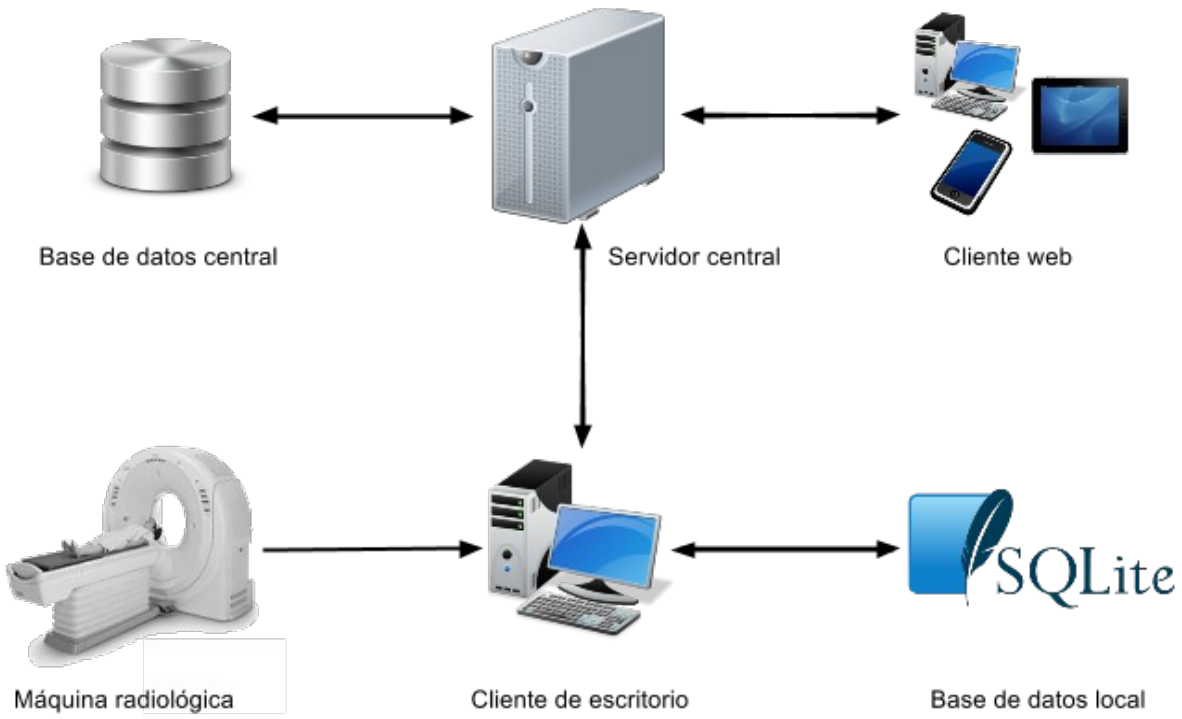

Figura 4.7: Arquitectura

4.2.2. Diagramas de clases

En este apartado se van a mostrar algunos diagramas relativos al diseño a nivel componentes. Esta etapa del diseño se centra en la organización de las clases e interfaces y cómo se relacionan entre sí. Debido a la complejidad del proyecto, no se pueden mostrar todos los diagramas de clases. Por ello, se mostrarán aquellos que tienen más importancia, ya sea porque se adaptan a un patrón de diseño predefinido o porque simplemente se crea conveniente mencionar.

4.2.2.1. Patrón MVC (Modelo Vista Controlador)

El patrón de diseño o arquitectónico Modelo Vista Controlador es el patrón que engloba toda la aplicación. Todos los componentes del sistema, se pueden ubicar dentro de uno de estos tres módulos. El objetivo de este patrón es separar la lógica de negocio, de la interfaz de usuario y los datos del modelo. A continuación, se definen los tres módulos que componen este patrón:

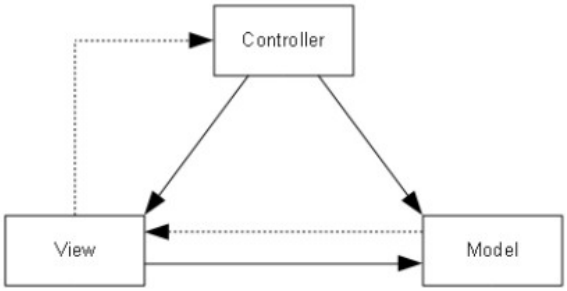

Figura 4.8: Modelo Vista Controlador

- **Modelo:** Este componente tiene que ver con las representaciones de la información que va a ser utilizada en el sistema. Un componente del modelo en el sistema es la clase Estudio, ya que es una representación de una entidad de información. La base de datos también se encuentra dentro de este componente.

- **Vista:** Es el componente con el que el usuario interactúa: la interfaz de usuario o GUI. En una implementación pura de MVC, la vista no tiene ningún estado y no realiza acciones, sino que las delega en el controlador.

- **Controlador:** Este componente es el encargado de gestionar la lógica de la aplicación, responder a eventos, peticiones de la vista por parte del usuario, etc. En este proyecto, el controlador se encarga de realizar las acciones solicitadas por la vista, además de observar constantemente el estado de la transferencia para reflejarlo la vista.

El uso de este patrón ha permitido afrontar de una manera mas sencilla los constantes cambios de la interfaz durante el desarrollo del proyecto. Gracias a tener la lógica separada de la vista se han podido realizar cambios en la interfaz sin necesidad de cambiar la lógica de la aplicación, puesto que solo se necesita modificar la representación de la información, no su tratamiento. Además toda la información de la vista se mantiene actualizada sin necesidad de que el usuario lo solicite manualmente.

4.2.2.2. Patrón Observador

En el patrón de diseño observador, un componente (observador) se suscribe a otro (observado) el cual notifica cuando el segundo cambia su estado, esto permite crear una cadena de notificaciones entre varios componentes automáticamente.

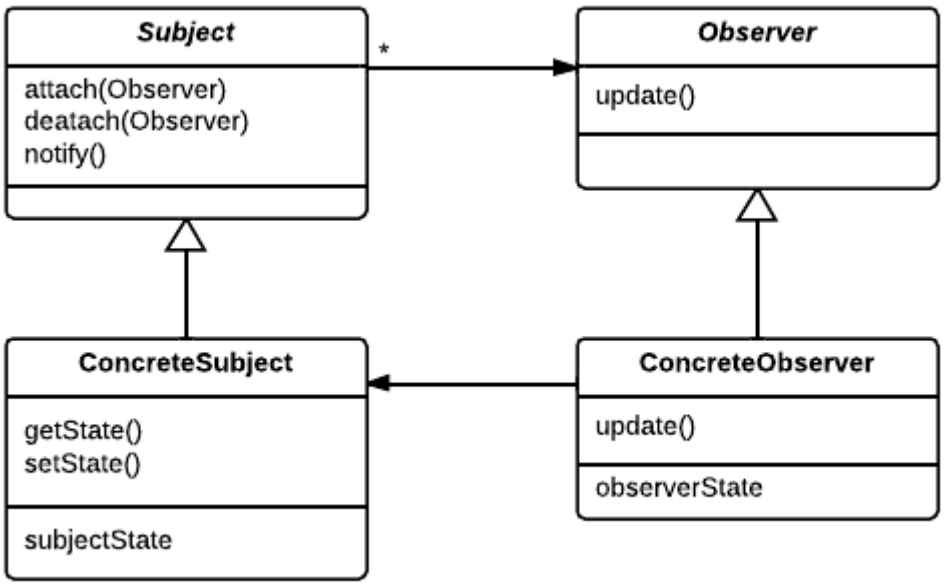

Figura 4.9: Patrón observador

Este patrón se ha utilizado en la interfaz y en la comunicación entre el controlador y el proceso encargado de escanear los estudios, utilizando los anteriormente mencionados *Signals* y *Slots*. El proyecto contiene una hebra en constante ejecución que escanea el directorio donde se almacenan los estudios cada dos segundos. El controlador se subscribe a esta hebra y en el momento que la hebra detecta que se han recibido o subido imágenes de cualquier estudio notifica al controlador a través de un *Signal* para que este realice las acciones necesarias, sin tener que comprobar constantemente el estado de los estudios.

4.2.2.3. Patrón Composite

El patrón Composite sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol. Esto simplifica el tratamiento de los objetos creados, ya que, al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

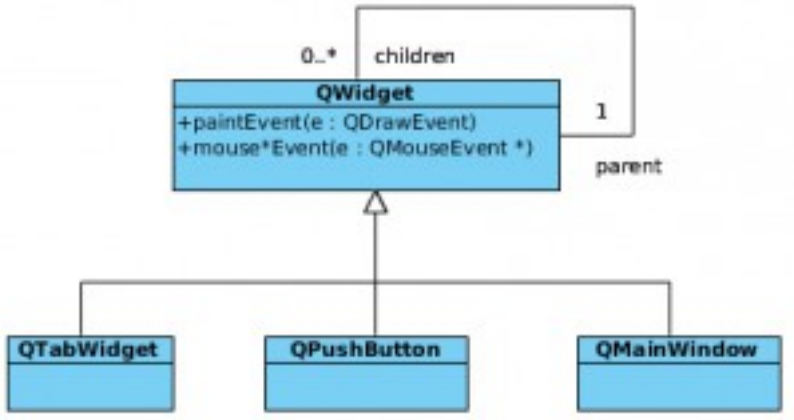

Figura 4.10: Patrón composite

Gracias a este patrón se puede construir y manejar componentes complejos de la interfaz gráfica de una manera más sencilla. En este proyecto ha sido utilizado para crear y agrupar de manera jerárquica componentes de la interfaz gráfica. Un ejemplo de estos componentes puede ser el menú de configuración, creado a partir de pestañas y diversos componentes más: Botones, entradas de texto, desplegables, etc.

4.2.3. Interfaz de usuario

Qt Creator ofrece la posibilidad de diseñar la interfaz mediante un editor gráfico para una interfaz estática o mediante código en *C++* para una interfaz dinámica. Independientemente de la forma en que se realice, cada pantalla se compone de un fichero XML con extensión *.ui*, que representa el aspecto gráfico de la interfaz y un fichero *.cpp* donde se desarrolla la lógica de la aplicación.

Para diseñar la interfaz se han consideraron los siguientes aspectos:

- **Interfaz familiar:** La mayoría personas pasa gran parte de tiempo visitando páginas web. Por ello se decidió diseñar una interfaz parecida a la de otra aplicación similar a la desarrollada como por ejemplo, el cliente de escritorio de *Dropbox*, para que le resulte al usuario mas familiar la navegación por la aplicación.

- **Consistencia:** La interfaz ha de mantener una consistencia entre sus diferentes ventanas, ya que una interfaz consistente permite al usuario entender mejor como funcionarán las cosas, incrementando su eficiencia.

- **Jerarquía visual:** Se ha diseñado la interfaz de una manera que permite al usuario centrarse en lo mas importante a la hora de realizar cada tarea. El tamaño, color y posicionamiento de cada elemento hacen mas sencillo entender la interfaz.

- **Proporcionar Feedback:** La interfaz ha de comunicarse con el usuario cuando sus acciones han sido realizadas de manera correcta, incorrecta o anda perdido.

- **Reflejar estado:** En todo momento la interfaz debe reflejar el estado en que se encuentra. No hay que obligar al usuario a recordar.

- **Diseño simple:** Se ha optado por un diseño simple sin muchos elementos innecesarios que distraigan la atención del usuario.

4.2.3.1. Login

La Figura 4.11 muestra la pantalla de *login*. El usuario que vaya a utilizar la aplicación debe estar dado de alta previamente en la base de datos central. Además, los datos de ese usuario estarán almacenados en la base de datos local, ya que la aplicación se ha de poder utilizar sin conexión a Internet. Por tanto, el proceso de *login* no se realiza contra el servidor, sino contra la base de datos del dispositivo.

Figura 4.11: Login

4.2.3.2. Transferencia

La Figura 4.12 corresponde a la pantalla de transferencias de la aplicación. Esta pantalla se muestra por defecto después que el usuario haya realizado el *login* correctamente. Cuenta con una serie de pestañas para poder navegar a las otras pantallas, además a la izquierda muestra el estado del servicio.

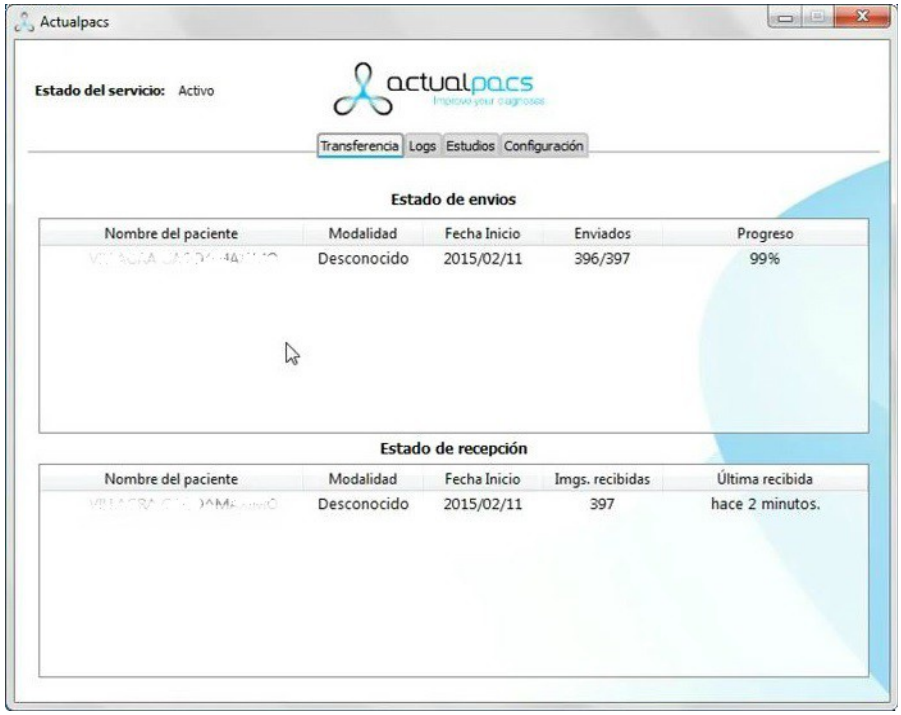

Figura 4.12: Transferencia

4.2.3.3. Logs

Si el usuario accede a la pestaña Logs, se mostrará la pantalla de la Figura 4.13. En esta pestaña se pueden observar todos los *logs* generados por el sistema, además de poder realizar un filtro para poder simplificar la información.

4.2.3.4. Estudios

Para realizar consultas a la base de datos local sobre los estudios y las instancias de cada estudio, el usuario debe acceder a la pestaña Estudios (Figura 4.14).

El usuario puede buscar un estudio determinado por nombre de paciente o identificador del estudio, además estos estudios pueden ser ordenados de manera descendente y ascendente por cada uno de sus campos. Para ver las instancias de cada estudio el usuario deberá pulsar sobre el estudio deseado.

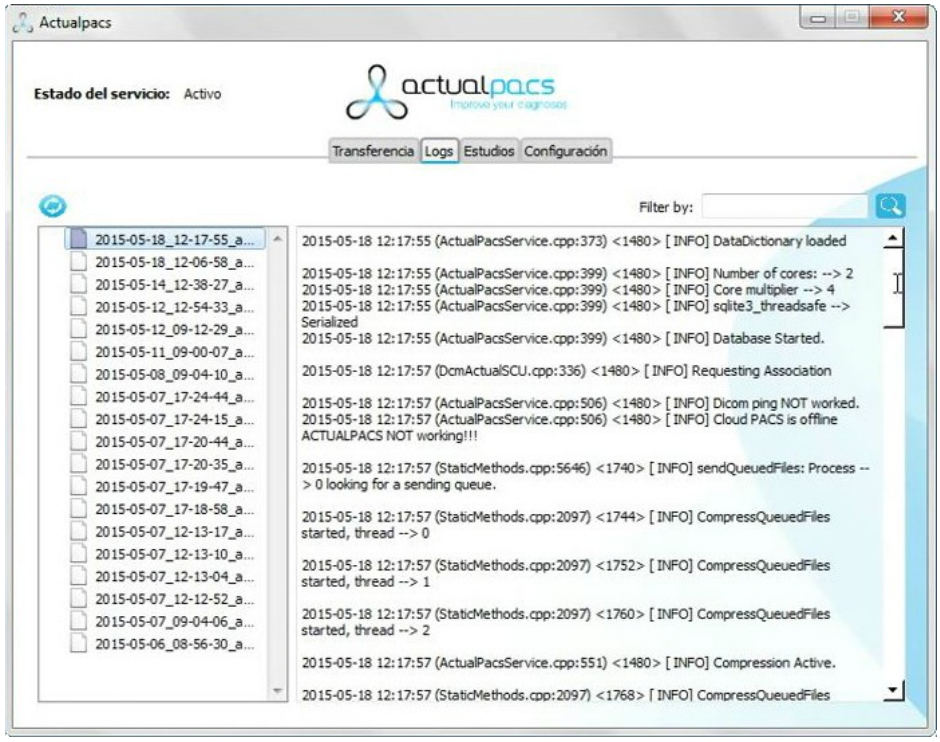

Figura 4.13: Logs

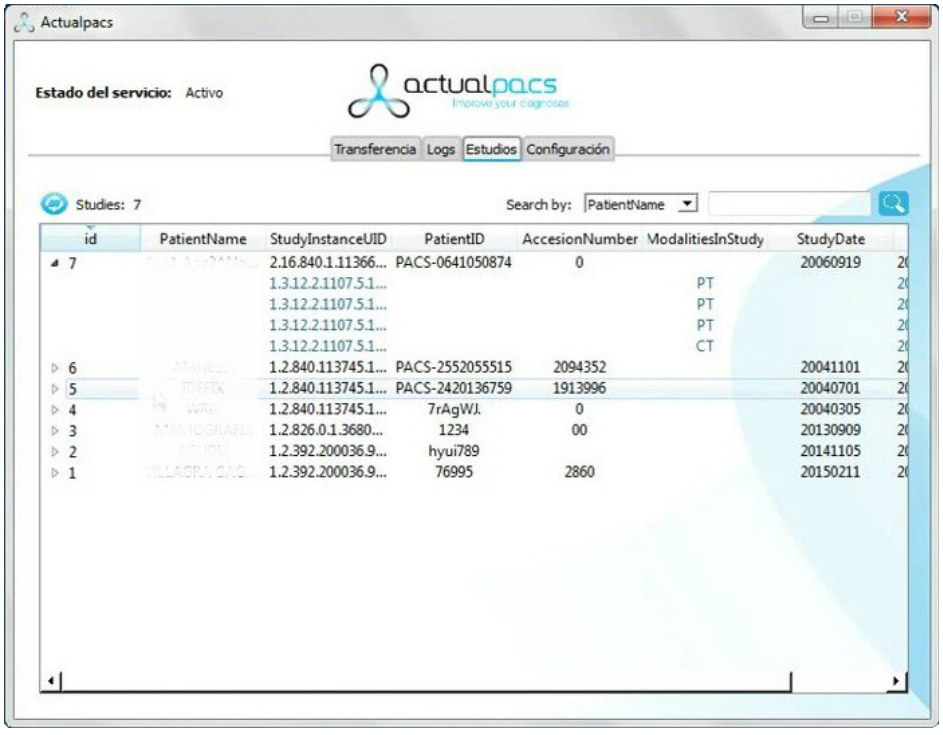

Figura 4.14: Estudios

4.2.3.5. Configuración

En esta pantalla se puede editar la configuración del sistema. En la Figura 4.15 se puede apreciar la pantalla de configuración básica y en la Figura 4.16 la configuración avanzada.

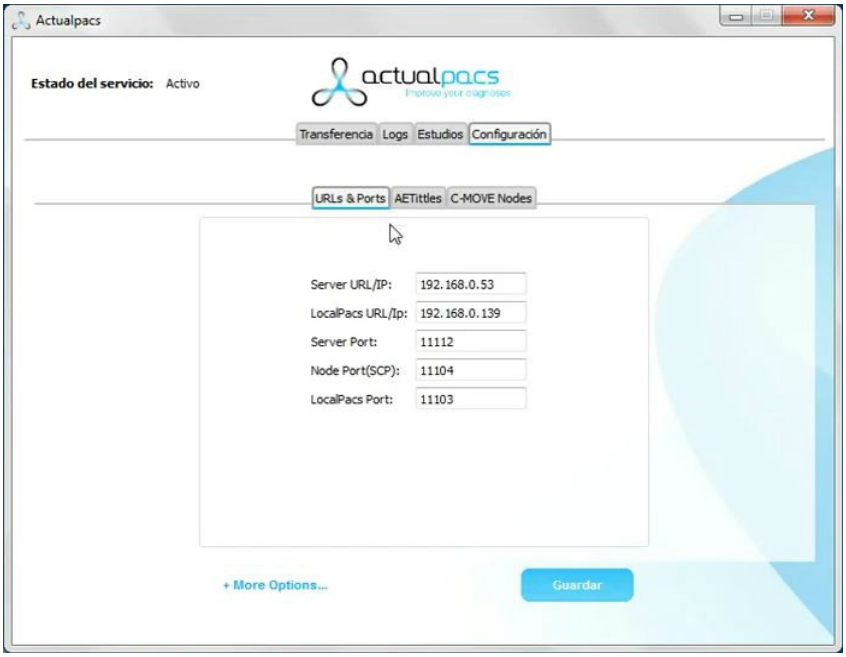

Figura 4.15: Configuración básica

Figura 4.16: Configuración avanzadas

La configuración del sistema se clasifica en las siguientes categorías:

- **URLs & Ports:** Ips/Urls y puertos de los componentes del sistema: *Pacs*, Servidor, etc.
- **AETitles:** Listado de las máquinas de las cuales se reciben las imágenes.
- **C-MOVE Nodes:** Parámetros necesarios para configurar el protocolo *C-MOVE* de DICOM.
- **Node DicomParams:** Parámetros DICOM varios del cliente.
- **Activation:** Varias opciones DICOM, las cuales pueden estar activas o inactivas.
- **Other:** Opciones más propias del sistema que del estándar DICOM.

Una vez se haya configurado el sistema, se debe pulsar el botón de guardar para aplicar los cambios. Una vez haya sucedido esto, debe saltar un mensaje de confirmación (Figura 4.17), y en el caso de aceptar se mostrará el progreso de los cambios aplicados (Figura 4.18).

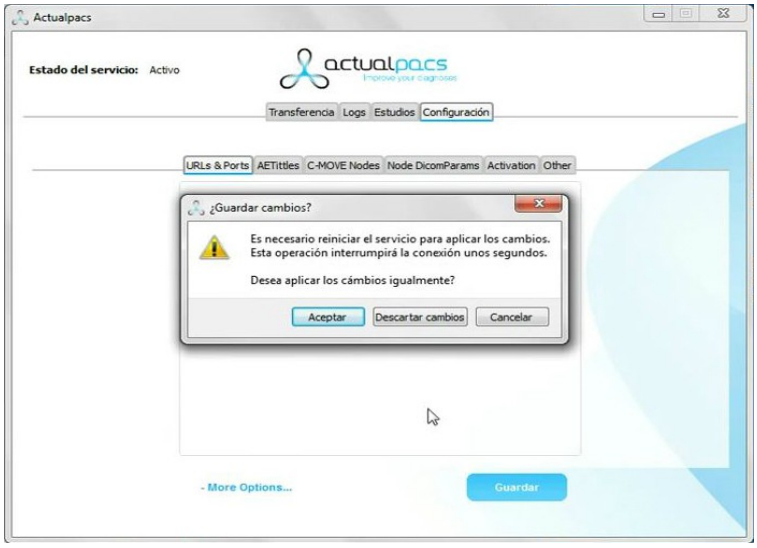

Figura 4.17: Mensaje de confirmación

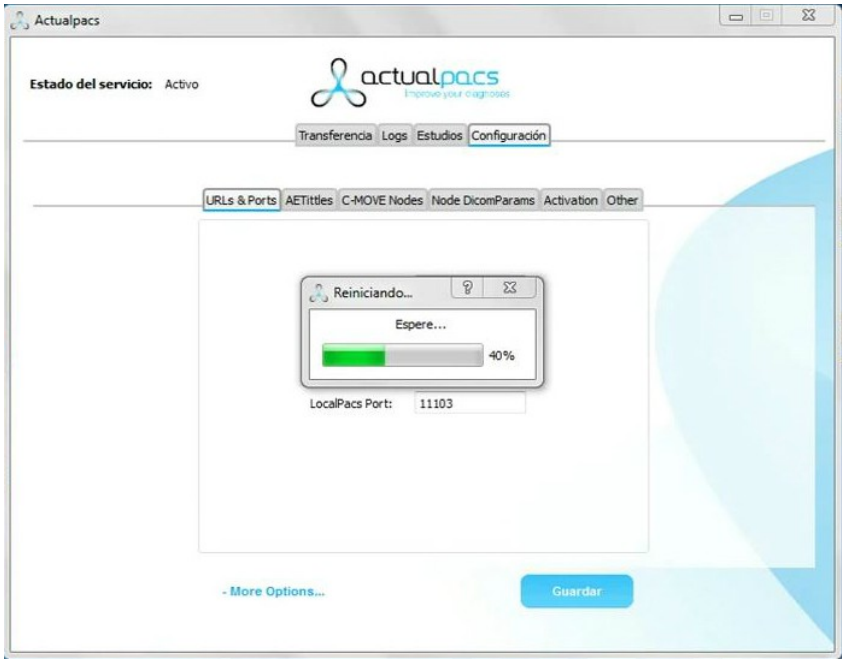

Figura 4.18: Progreso

4.3. Validación y verificación

El objetivo de la fase de validación y verificación del sistema es comprobar el funcionamiento de este a todos los niveles. Esta fase es esencial en el proceso de desarrollo de software ya que proporciona información sobre la calidad del sistema que se está desarrollando. Existen varios tipos de pruebas según su alcance o según sea el tipo de componente que se está probando:

- Pruebas unitarias
- Pruebas de usabilidad
- Pruebas de rendimiento
- Pruebas de aceptación

El objetivo de todo desarrollo software no es llevar a cabo todos los tipos de pruebas, sino que para cada caso concreto hay que realizar un estudio con el objetivo de definir qué pruebas son útiles y viables, teniendo en cuenta variables como el tiempo o el presupuesto.

En este proyecto se han realizado pruebas unitarias, de rendimiento y sobretodo de usabilidad, ya que se está desarrollando una interfaz gráfica y este es uno de los aspectos más importantes que hay que tener en cuenta. Además se han seguido las reglas heurísticas propuestos por Nielsen (1994, p.152-158) para garantizar una interfaz de calidad del agrado del cliente.

4.3.1. Pruebas unitarias

Las pruebas unitarias son aquellas que se encargan de comprobar el correcto funcionamiento de cada módulo o componente que constituye el sistema. La prueba de módulo es independiente de los demás, de esta forma nos aseguramos de que cada uno de los módulos trabaja como se espera por separado.

Para desarrollar las pruebas unitarias en este proyecto se ha utilizado la librería *QtTest*, la cual permite realizar tests tanto de la lógica interna como de la interfaz gráfica.

Las pruebas se realizan en bancos de prueba. El banco de pruebas está representado por una clase *QObject* siendo los tests funciones s*lot*. Hay una convención de nombres especiales que se utilizan en los test y se explican a continuación:

- Cada prueba debe tener el prefijo test en el nombre.
- *initTestCase()* y *cleanupTestCase()* son métodos llamados antes y después de la ejecución del banco de pruebas.
- *init()* y *cleanup()* son métodos llamados antes y después de cada test.

La Figura 4.19 muestra un ejemplo de una clase test sin implementar sus métodos.

Cualquier función cuyo nombre no siga el convenio establecido por *QtTest* es ignorada y no se ejecuta como test. Por este motivo se puede crear funciones auxiliares para los test y no serán ejecutadas como tal.

```
#include <QtTest>

class TestEjemplo : public QObject {
    Q_OBJECT

private slots:
    // funciones ejecutadas por QtTest antes y despues del
       banco de pruebas

    void initTestCase();
    void cleanupTestCase();

    // funciones ejecutadas por QtTest antes y despues de
       cada test

    void init();
    void cleanup();

    // pruebas unitarias - todas las funciones con prefijo "
       test" seran ejecutadas como tests

    void testSomething();
};
```

Figura 4.19: Clase QtTest ejemplo

Para confirmar que una función ha pasado las pruebas correctamente *Qt* utiliza la función *Qverify*, la cual recibe como parámetro un *booleano*. La Figura 4.20 muestra un ejemplo de ello.

```
void TestEjemplo::testSomething() {

    int studyFolders = getStudyFolders('123456');

    if(studyFolders==6)
        QVERIFY(true);
    else
        QVERIFY(false);
}
```

Figura 4.20: Clase QtTest implementación

Una vez se han creado los bancos de pruebas, para que *Qt* pueda ejecutarlos se han de incluir en el método *main* de la aplicación. En la Figura 4.21 se puede ver el método *main* ejecutando varios bancos de prueba de ejemplo.

```
#include <QtTest>
#include "testejemplo.h"

int main(int argc, char** argv) {
    QApplication app(argc, argv);

    TestEjemplo testEjemplo;

    return QTest::qExec(&testEjemplo, argc, argv);
}
```

Figura 4.21: Ejecución banco de pruebas

Después de ejecutar los bancos de pruebas, los resultados pueden verse directamente por la consola de *Qt* o por el contrario, si se desea, pueden exportarse a un fichero de texto.

4.3.2. Pruebas de rendimiento

Las pruebas de rendimiento sirven para determinar las velocidad en la que se realiza una tarea determinada del sistema en condiciones particulares de trabajo. También sirven para validar otros factores de la calidad del sistema, como son la fiabilidad y uso de recursos.

En este proyecto se han realizado pruebas de estrés y de carga, para determinar cuántos estudios es capaz de analizar el sistema y corregir cuellos de botella.

En un entorno con 1.000 estudios y una media de 100 imágenes cada uno, el sistema se bloqueaba, daba error y se cerraba repentinamente. Sin embargo después de realizar estas pruebas se detectaron cuellos de botella generados por un gran numero de consultas a la base de datos, un mal diseño del algoritmo de escaneo de estudios y gran cantidad de recursos gráficos en la interfaz.

Para solucionar estos problemas, se corrigieron algunos errores en el algoritmo de escaneo de estudios, el cual hace un recorrido iterativo del sistema de archivos donde se almacenan los estudios. De esta manera comprueba si han habido modificaciones recientes para detectar nuevos estudios y actualizar el estado de la transferencia de los estudios existentes. También se utilizó memorización, es decir, en vez de acceder a la base de datos cada iteración del algoritmo para obtener información necesaria de

los estudios, se creó una estructura de datos que los almacenaba en memoria y así se evita un gran numero de accesos a la base de datos.

Con esto y remplazando elementos gráficos costosos de procesar, por otro mas simples, se solucionaron estos problemas. Actualmente el sistema es capaz de analizar 6000 estudios con 100 imágenes de promedio cada uno, en 2 segundos. Esto supone que el sistema puede soportar una carga de trabajo 10 veces superior a la carga recibida en un entorno normal.

4.3.3. Pruebas de usuario

Las pruebas de usuario se basan en la observación de cómo un grupo de usuarios lleva a cabo una serie concreta de tareas encomendadas por el evaluador, analizando los problemas de usabilidad que se encuentran.

Aún cuando el diseñador tenga amplios conocimientos sobre usabilidad, resulta recomendable evaluar el diseño con usuarios. Esto se debe a que, conforme más tiempo dedica un diseñador a un proyecto, menor es su perspectiva y más difícilmente detectará posibles problemas.

Podemos decir que gran parte de lo que el diseñador percibe cuando mira su propia obra, es una construcción mental; ve aquello que tienen en mente, no aquello que sus usuarios tendrán ante sus ojos.

El numero de participantes recomendados para la prueba son 15, pero en este proyecto se ha realizado la prueba con 4 usuarios, todos ellos empleados de la empresa pero ajenos al proyecto. Cada usuario, ha realizado sobre prototipos de alta fidelidad una serie de tareas típicas que un usuario real llevaría a cabo: Cambiar el puerto SCP en la configuración, cambiar el idioma de la aplicación, buscar un estudio concreto en la base de datos, filtrar errores en los *logs*, apagar y encender el sistema.

A medida que se han ido realizando estas pruebas, se han encontrado errores como la falta de ayuda en la interfaz, escasa prevención de errores, gran número de pasos para realizar algunas acciones y uso de terminología técnica en varios mensajes.

Para solucionar la escasa prevención de errores se ha deshabilitado funcionalidades (como la gestión de certificados), las cuales pueden conllevar a errores y no se consideran necesarias para el usuario. También se han añadido *tooltips* en varios elementos de la aplicación para que el usuario situando el ratón encima del elemento durante unos segundos reciba información sobre este. Respecto al uso de terminología técnica y gran numero de pasos, se han solucionado reescribiendo los mensajes de la aplicación y re-diseñando la secuencia de pasos de algunos procedimientos, utilizando otros elementos de la interfaz mas potentes que la acortaban.

Gracias a las pruebas realizadas a lo largo del proyecto y a los prototipos de la interfaz, se han detectado y corregido estos errores, lo cual ha favorecido al desarrollo del proyecto y a alcanzar los objetivos antes del tiempo estimado.

Capítulo 5

Conclusiones

A lo largo de este trabajo de fin de grado se ha desarrollo una aplicación de escritorio para el sistema operativo Windows y OSX enfocada al campo de la radiología e imagen médica.

Este sistema se ha desarrollado satisfactoriamente, cumpliendo con todos los objetivos y requisitos propuestos al inicio de la estáncia. En la actualidad se encuentra funcionado en entorno real, permitiendo a los usuarios de *Actualpacs* visualizar la transferencia estudios radiológicos a la nube y llevar a cabo el mantenimiento de estos siguiendo el estándar DICOM.

Antes de iniciar el desarrollo de la aplicación, ha sido necesario aprender una serie de tecnologías y adquirir una serie de conocimientos sobre los estándares de imagen médica. A parte de la implementación propiamente dicha, se ha realizado una fase de análisis, ya que los requisitos de usuario y la lógica del sistema no estaban muy bien definidos al principio de la estancia en la empresa. Además, puesto que lo mas importante de la aplicación es la parte gráfica, ha sido necesario profundizar en aspectos de diseño, componentes gráficos y usabilidad.

Atendiendo al desarrollo del proyecto, al principio fue difícil y confuso sintetizar la idea del proyecto, ya que no estaban bien definidos sus requisitos y no estaba claro que se quería. Una vez realizada la fase de análisis y diseño, todas estas dudas desaparecieron. Esto ha servido para valorar la importancia de tener bien definidos los requisitos de un proyecto al inicio de su desarrollo.

La empresa ha propuesto para un futuro varias mejoras y funcionalidades para el proyecto. Entre estas tareas se encuentra la posibilidad de que el sistema busque actualizaciones y se actualice de manera automática.

Finalmente, además de haber aprendido a desarrollar un proyecto de principio a fin, la realización de este proyecto me ha proporcionado conocimientos sobre una serie de tecnologías que desconocía, incluyendo temas relacionados con la radiología, imagen medica y el estándar DICOM. Además, se me ha permitido poner en práctica, en un entorno empresarial real, muchos conocimientos adquiridos durante la carrera, tales como la metodología agil SCRUM, patrones de diseño, evaluación de interfaces, *testing*, etc.

Bibliografía

[1] J. Blanchette, M. Summerfield. C++ GUI Programming with Qt4. Prentice Hall; 2nd edition (2008).

[2] E. Freeman, B. Bates, K. Sierra. Head first Design patterns. O'Reilly Media; 1st edition (2004).

[3] Jay A. Kreibich. Using sqlite. O'Reilly Media; 1st edition (2010).

[4] Roger S. Pressman. Software engineering: A practitioner's approach. McGraw-Hill Higher Education; 8th edition (2014)

[5] Nielsen, J. (1994). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY